Java 103

Lesson 1

abstract "مجرد/تجريدي" أضيفت لاحقاً

* يتم كتابتها قبل كلمة class مثل abstract

Ex: public abstract class First {

}

* معنى إنك تضع abstract قبل الكلاس ⇐ لا تريد إنشاء كائنات من هذا الكلاس

* abstract و new لا يتفقان مع بعضهما أي لا يمكن إنشاء كائن (جديد) من الكلاس فانتبه لذلك.

* فائدة هذا الكلاس الذي يكون قبله abstract ⇐ يستفاد منه لأغراض الوراثة وتعدد الأشكال polymorphism

لأنه لا يمكن إنشاء كائن منه.

Lesson 2

معلومات إضافية عن abstract

* الكلاس الذي فيه abstract بالإمكان استخدامه أيضاً: لجميع الدوال والمتغيرات المشتركة بين جميع الكلاسات التي ستورث من هذا الكلاس.

Lesson 3

شرح مفهوم abstract Method

* abstract Method تعتبر جزء تكاملية تضيفها لـ abstract class

* الكلاس العادي (الي من غير abstract) يسمى Concrete class ← يعني كلاس أقدر أنشئ منه كائنات عكس الـ abstract ليست كلمة في البرمجة بس.

* أنواع الكلاسات ← Concrete class
abstract class

* الدالة أو الـ Method التي تكون abstract تكون بلا جسم يعني بلا أقواس {} ووضع آخر الدالة ; مثال: public abstract void print(); ← يعني دالة غير مطبقة غير منفذة ما لها كود.

* عند إنشاء Abstract Method: يجب أن تكون بداخل كلاس نوعه abstract

* معنى Abstract Method :- هي دالة يتم وضعها داخل abstract class ويُترك تنفيذها أو كتابتها للكلاسات التي ستورث من الكلاس الي فيه abstract "يعني كأنه ضيفل override".

* القاعدة ببساطة هي أن جميع الدوال التي تكون من نوع abstract سيتم تطبيقها وكتابتها في أي كلاس يرث من abstract_class First .

* الآن في الكلاس الـ Concrete والذي يرث من الـ abstract في المثال سيأتي لهفاً وهو إنه يجب عمل override للدالة الـ abstract حتى نقدر نتعامل معها.

* فعند إعادة كتابة الدالة الـ abstract في الكلاس الـ Concrete سيتم كتابة الدالة عادي بدون كلمة abstract ونضع @override . وتكتب الدالة بالصيغة العادية.

Lesson 4

مثال توضيحي لمفهوم Abstract
وكلاس shap

شاهد الفيديو ^^

Lesson 5

استخدام abstract مع Constructor, static

* لا يمكن استخدام abstract مع static . XXX public static Abstract ---

* الـ Constructor لا يورث وإنما يُستدعى من الكلاسات الموروثة

* لا يمكن استخدام Abstract مع الـ Constructor

* الخلاصة: تذكر أن

لا يمكنك وضع abstract مع static أو مع Constructor

---

lesson 6

توضيح استخدام abstract
في مفهوم تعدد الأشكال

* جاء أن مثلاً إنت عملت كلاس Abstract وبرث منه ٣ كلاسات مثلاً.

الآن لو رحت لأي Concrete class وكتبت فيه دالة بالشكل التالي :- (Abstract class)

```
public void drawshap(shap s)
{

}
```

لاحظ هنا تم قبول الجملة لأنه لم نستخدم new " يعني المفروض يرفض لأن shap نوعه abstract

⇐ ولكن هنا قوة الـ polymorphism تكون واضحة حيث تم قبول الكود

لأنه يقصد بـ s shap ⇐ إنه هيمرر كائن من (يشير إلى) أحد الكلاسات التي ورثت s shap . يعني مستحيل يمرر shap لأنه Abstract

⇐ s لازم يكون أحد الكائنات من الكلاسات الموروثة من shap .

---

lesson 7

توضيح استخدام abstract
في مفهوم تعدد الأشكال

Ex

```
public shape getshap()
{
circle c1 =new circle();
return c;
}
```

* نوع الإرجاع نوعه shape ولكن (زي void, int)
ليس shape نفسه بل كائن من كلاس مشتق من shape (كائن من)

* الآن أصبحت الدالة getshap تطبق مفهوم الـ polymorphism في الإرجاع

```
circle C = new circle();
Rectagle R = new Rectagle();

shap S = C;
```

* لم يعطي خطأ هنا لأننا لم نعمل new لـ s shap
لكن هو s أخذ قيمة " بعمل ربط بين الإنشاء وأخذ القيمة"

* يعني كأنها كائن من shape " اللي ميتفعش " يمثل كل اللي ينفع
(بعمل عمله الـ polymorphism)

s تأخذ أكثر من شكل ممكن دائرة C أو مستطيل R

لم ننشأ كائن لأنه معملش new

Lesson 8

استخدام abstract مع مفهوم تعدد الأشكال polymorphism

Rec r = new Rec();

* كده معناها إنك عملت كائن من shap ; r = s ; Shap s

* بل معناها إنه كائن بداخله كائن لأوامر الكلاسات المشتقة من shap يشير له

if (s instanceof Rectangle) // يعني لو s نوعه Rectangle

نسخة من الكائن

{

المثال في الفيديو

}

Lesson 9

شرح الـ interface وبعض implements

* interface: هي طريقة للتواصل بين الكائنات بصفة عامة.

* امسح كلمة class واكتب interface

public class MyInterface

interface

* الـ interface كأنك بتكتب كلاس عادي، لكن الفرق إنه يحتوي على دوال بدون Body

كل الدوال جميعها لا تحتوي على Body

يعني ليست منفذة ليست implemented ← زي الـ abstract، لكن بدون كتابة abstract.

Ex: public void start();

* الدوال المكتوبة بداخل كلاس interface تكون public بشكل افتراضي حتى لو لم تكتبي.

* لا يمكن إنشاء كائن من الـ interface لأنه يُعامل معاملة الـ Abstract.

* تحتوي على ثوابت final عادي جدًا

* الـ interface يحتوي على قائمة من الدوال الغير منفذة.

* كيف سيتم تنفيذ الدوال فيه بما أنه الدوال كلها بدون Body !؟

إذن عن طريق كلاس آخر يعمل implementation لهذا الـ interface.

* في كلاس التنفيذ يتم كتابة هكذا: public class App implements MyInterface {...}

* معنى هذا الكلام: App كلاس عادي وهينفذ جميع الدوال الموجودة في الـ interface اللي اسمها MyInterface

* ممكن نكتب دوال جديدة، لكن يجب عمل override للدوال الموجودة في الـ interface كلها

lesson 10

معلومات إضافية عن Interface

* الـ interface تربط بين عدة كلاسات عن طريق الوراثة ! .

* أي كلاس يعمل implements لـ interface فلازم يحتوي على الدوال الموجودة في الـ interface كلها.

lesson 11

تمرير الـ interface كباراميتر

```
public void printVal(MyInterface in)
{
    in.printData();
}
```

* عرفت كائن نوعه interface اسمها MyInterface
* ممكن نعمل كائن من الـ interface لكن ده كائن يشير لأي كائنات الكلاسات اللي هتنفذ الـ interface (implements)

* تعرف الـ interface في الباراميتر عادي جدًا مثل أي كلاس

ولكن لا تقدر تستخدم معها new لأنها أساسًا

لا يوجد بها دوال

⇐ المعنى إنك تبعت في الدالة باراميتر نوعه interface : يعني تقدر تمرر أي كلاس يطبق هذه الـ interface (كائن)

* ميزة الـ interface : إنك بتستخدم الدوال الموجودة في كلاس interface بدون معرفة إيه اللي هيتنفذ

* فالمعرفة اللي سيتم تنفيذه يعتمد على الكلاس الذي سيتم تمريره .

مثال على استخدام الـ interface

lesson 12

* وعندك برنامج مثلاً ويقبل plug-in "إضافات" وتمرر له الإضافات للمطورين

مثلاً مطور هيضيف إضافة الـ pdf - تشغيل الصوت - ... الخ فكيف تدير هذا الكود لأن المطور

هيكتب الكود بتاعه هنا بقا هتستفيد من الـ interface ممكن تعمل كلاس plugin وهو interface

وأي شخص هيعمل إضافة لازم يطبق هذه الـ interface بكل دوالها .

* لكن لن تعرف ما المكتوب داخل الدوال لأن المطور هو اللي هيكتبها بس هو ملزم بالدوال هذه.

Lesson 13

استخدام أكثر من interface
Implement Multi-Interfaces

Ex: public class Demo implements X, Y, Z { }

حيث X, Y, Z عبارة عن interfaces.

⇐ المعنى كده أن جميع الدوال X, Y, Z لازم تُنفذ بداخل الكلاس Demo بثلاث

⇐ تم عمل هذه الحركة حل مشكلة Multi_Inheretance " C++ تدعم تعدد الوراثة

بينما جافا لا تدعم " لكن بهذه الحركة حيلة.

⇐ هام: حسب الفيديو " Demo بتنفيذ demo " أكثر من كده " Demo, Z, Y, X "

Lesson 14

عمل الـ interface لـ extends

* هام شاهد الفيديو

Ex: public interface MyInterface2 extends MyInterface
{ public void justLoad();
}

الـ interface مع الـ interface بيعمل وراثة.

⇐ الـ interface مع الـ interface extends interface.

انتبه ⇐ عندما نطبق interface مع كلاس نقول implements.

⇐ الكلاس مع الكلاس extends.

~~⇐ الـ interface مع الكلاس extends.~~

⇐ الكلاس مع الـ interface implements interface

* الآن لو الـ interface التي تسمى MyInterface2 جميع دوالها بكلاس ⇐ MyInterface من 2
MyInterface2 من 1 ⇐

Lesson 15 — extends و implements

```
public class Adv extends First implements MyInterface, plugin
{
}
```

* بالإمكان إنك تطبق الوراثة و الـ interfaces الـ implements في سطر واحد

* الآن الكلاس Adv يرث من First و يطبق الـ MyInterface و كمان plugin

```
Advance a = new advanced();
```

* الآن الكائن a نوعه Advance بين نوعه First لأنه يرث منه، و نوعه MyInterface، و كمان نوعه plugin « مفهوم عام »

* يعني a كائن يرث من First و يطبق الاثنين الـ interfaces

Lesson 16 — Anonymous Class و استخدامه مع الـ interface

* طبعاً نادراً ما يستخدم في الأنترويد نت و قواعد البيانات.

Ex
```
First f = new First();
App a = new App();
f.printVal(a);
```

a من App و بما أنه ينفذ الـ MyInterface فسيتم تمريره عادي لأن printVal في MyInterface بيستقبل

output : welcome to app

```
First f = new First();
MyInterface i = new MyInterface(){ الدوال التي سيتم تنفيذها في الكلاس الذي يطبق الـ MyInterface };   ← : syntax
```

* و هذا ليس معناه إنك عملت كائن من MyInterface

* بالإمكان عمل تطبيق لـ interface بهذه الصيغة بدلاً من إنشاء كلاس جديد و عمل implements.

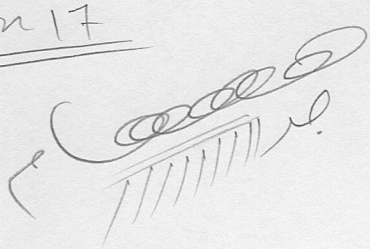
* معنى الـ syntax : أنشئ كائن جديد من هذا الكلاس "يجب أن يكون الكلاس يطبق الـ interface" و يلزم يتم وضع دوال الـ interface بداخل الأقواس {=}

* التعبير: كأنك عملت كلاس جديد بدون اسم «Anonymous Class» وأنشأت منه كائن.

* شاهد الفيديو هام جدًا

---

lesson 17



كلاس Anonymous object

هامة جدًا جدًا جدًا في الأندرويد والـ Gui

* كما في الدرس السابق الكلاس اللي من غير اسم تكوينه الـ syntax كما يلي:

```
MyInterface i = new MyInterface(){ ≡ };
```

① الكلاس الوهمي ② انشاء منه كائن ③ يخزن في i

كلاس جديد بدون اسم وانشأنا منه كائن

④ يتم استدعاء الدالة العلوية ومرر i مثل: `F.printVal(i);`

كائن من كلاس First تم تعريفه

دالة تأخذ باراميتر نوع MyInterface

الجديد:

* جافا تقبل sytax معقدة جدًا

* الـ syntax الجديد أن أثناء تمرير للباراميتر `F.printVal(( ));`

يقبل أي كائن من MyInterface

Syntax:

```
F.printVal( new MyInterface(){ ≡ } );
```

كلاس بدون اسم ح كائن بدون اسم

الدوال التي سينفذها الكلاس الذي سيطبق MyInterface implements

⇐ هام جدًا هنتعامل معاه بكثرة في الأندرويد والـ Gui.

lesson 18 — الكلاسات و Anonymous object

* هنا أنشئت كائن بدون اسم
وكذلك كلاس بدون اسم

- `new MyInterface() {=};`
  interface الذي أورث

Example in video :-

```
new MyInterface(){
    public void printData(){ System.out.println("aman") ;}
}.printData() ;
```

↑ لاستخدام هذا الكلاس أو الكائن في نهاية {} الدالة يتم وضع نقطة ثم اسم الدالة

الشرح :- أنشئ كائن بدون اسم من الكلاس الذي بدون اسم ثم بعد الإنشاء
استدعي منه دالة printData

---

lesson 19 — تحويل الأنواع "Type Casting"

* يعني تحول من نوع بيانات وال آخر

```
int x = 10 ;
int y = 20.44 ;

int z = x ;   ← int نوعهم x و z عشان
int z = y ;   ✗✗ خطأ عشان مختلفين في النوع
int z = (int) y ;   ← صيغة التحويل
```

int ← عشان أحول لـ int
(int) ← عشان أحول الـ y

← يعني خذ قيمة y وحولها int وخزنها في z

* يمكن استخدام التحويل أيضاً مع الكائنات

* التحويل للكائنات سنراه بالتفصيل في الفيديو ، لاحقاً

lesson 20

معلومات إضافية عن Type casting

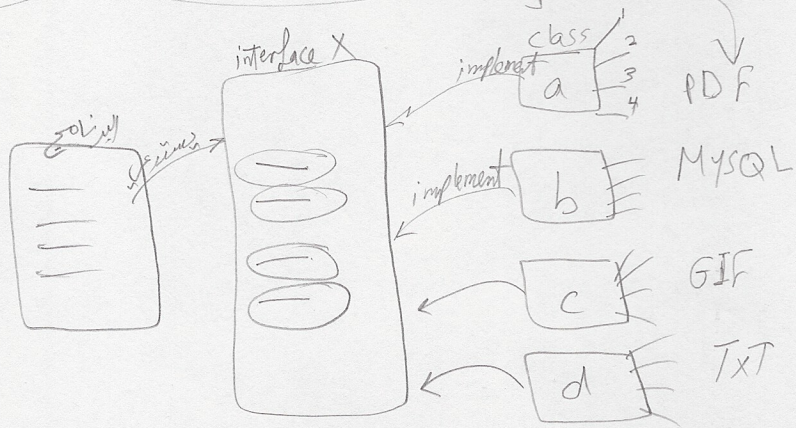
+ شاهد الفيديو

---

lesson 21

شرح تفصيلي لـ Interface

كل واحد من هذه الكلاسات يطبق الـ interface بطريقة مختلفة لكن مع وجود نفس الدوال الأربعة.



---

lesson 22

شرح الـ Empty Interface

* لا يستخدم الكلاسات التي تطبق الـ interface في كل مكان يتم فيه Empty Interface
وهذه الكلاسات المطلوبة تـ implements الـ interface . ومن ثم نمرر لأي كلاس
عادي ونعمل دالة مثلاً :

```
public void (Interface i)
{
}
```

هنا البرامتر هو أي كائن
من الكلاسات التي تطبق الـ interface

EX2:

```
public EInterface getIt()
{
    A z = new A();
}   return z;
```

EInterface يرجع كائن من نوع كلاس يطبق
وهو A

lesson 23

مثال لشرح فائدة الـ interface الـ plugin

هام

---

lesson 24

ما هو enum

* اختصاراً لـ enumeration "التعداد"

+ التعريف: هي عبارة عن طريقة لجمع ثوابت لها علاقة ببعضها البعض في شيء واحد
  - تقدر تستخدمها في أي مكان
  - الثوابت تكون محددة بعدد معين

* enum: هي طريقة تستخدمها إنك تضع بداخلها ثوابت.
وهذه الثوابت تستخدمها في أي مكان وهذه الثوابت تكون محددة بعدد معين
ومثلاً لها علاقة ببعضها البعض مثال: أيام الأسبوع = ٧ أيام ثابتة مفيش غيرها.
ولها أسماء محددة - مثال آخر: الجنس: ذكر/أنثى ---

```
package x;
public enum Gender {
    MALE, FEMALE;
}
```

أنشأت enum
وضع بداخله الثوابت
والثوابت كالعادة نعرف برمجي تكتب بحروف كبيرة

⇐ الآن أنشأت ثابتين MALE و FEMALE

⇐ أي ثوابت بداخل enum تعتبر static و final افتراضياً ولا يمكن تغييرها

```
system.out.println(Gender.MALE);
```

output: MALE

+ enum: أي حاجة تتحدد ومحددة ⇐ أيام الأسبوع - الجنس - شهور السنة ....

* بإمكانه كتابة متغيرات ودوال عادية داخل enum زي الكلاس.

lesson 25 — استخدام الـ enum مع parameters

* الثوابت يفصل ما بينها فاصلة عادية , وفي الآخر فاصلة منقوطة ;
* قيم الثوابت توضع بين الأقواس
Ex: MALE (10) , FEMALE (20) ;
* القيم كلها لازم تكون من نوع واحد int مثلاً

private int val;

private Gender(int x)
{ this.val = x;
}

* القاعدة: إذا كان للثابت قيمة ⇐ لازم تعرف له متغير داخل الـ enum
* هذا المتغير هيحتفظ بالقيمة اللي بين الأقواس .
* وبعدد الثوابت هيكون هناك نسخ من المتغير val

يعني هنا مثلاً ثابتين ⇐ لما تستدعي الثابت الأول هتخزن القيمة ١٠ في val
ولما يستدعى الثابت الثاني هيكون له نسخة خاصة من val وهيخزن فيها ٢٠
يعني كل واحد منهم هيكون له نسخة val خاصة فيه

* الـ enum لا تقدر تنشأ من كائن لأن الإنشاء أوتوماتيك ← Gender. إلخ
* يجب عمل Constractor داخل enum لو هتضع قيم للثوابت ويكون هذا الـ Constractor private ⇐ والسبب
* البارامتر الخاص بالـ Constractor لازم يتوافق مع النوع الذي تم وضع القيم فيه بتاعة الثوابت
* الآن لو أنت استدعيت MALE(10) ⇐ كأنك استدعيت الـ Constractor بـ ١٠
* بالإمكان وضع أكثر من قيمة مثل MALE(10, "male Aram") وكمان هنا يجب إنشاء متغيرين واحد نوعه int والآخر string , كذلك في حالة الاستدعاء والإنشاء لـ Constractor .
* في حالة وضع المتغيرات private يجب عمل دوال الـ Getter & setter .

lesson 26 — استخدام الـ Enumerator

* طريقة استدعاء القيم من الثوابت

اسم الكلاس
Gender.MALE.sVal
↳ الثابت
↳ القيمة المخزنة داخل MALE

* enum طريقة مريحة لعمل الثوابت
* شاهد الفيديو للتوضيح

---

lesson 27 — استخدام الـ Enumerator بداخل الكلاس

* بالإمكان كتابة enum بداخل الكلاس، ويعتبر جزء من أجزاءه.

```
public class A {
    public int x;
    public enum Gender {
        MALE, FEMALE;
    }
}
```

* وزي الدرس السابق بالضبط

---

lesson 28 — استخدام الـ Enum مع الـ Method

* أي ثابت يتم تعريفه مثلاً: MALE(10,"Hi") — يعتبر كأنه كائن نوعه Gender مطابق لصيغة التسمية.

* وبما أنه كائن فمعناها أنه يمتلك جميع الأشياء التي تعرفها داخل enum

* يعني أي شيء تعرفه داخل الـ enum ⟸ يمتلكه كل ثابت

* يعني ما تكتب أي دالة يمكن استدعائها عادي تعريف كلاس Gender داخل enum وبه ثابت MALE ودالة print

∴ Gender.MALE.print();

lesson 29

استخدام enum مع Abstract

* ال enum يقبل بداخله دوال Abstract "Abstract Method"

* أي دالة abstract موجودة بداخل enum ⇐ لازم كل ثابت موجود بداخل enum يعمل لها @override

```
public enum Gender {
    MALE(10,"welcom")
    { @override
      public void printString()
      { system.out.println("Male");
      }
    }
```

تكملة الكود (المتغيرات و Constractor)

```
public abstract void printString();
```

---

lesson 30

استخدام enum مع switch

* من مميزات enum إنك تقدر تستخدمها مع switch

Gender x = Gender.FEMALE ⟶ يعني x قيمته FEMALE

```
switch(X)
{ case Female:
      system.out.print"Hi";
      break;
}
```

* يعطي البرنامج ميزة القابلية للقراءة والترتيب

---

lesson 31

استخدام enum مع مصفوفة values

* لو عندك كلاس اسمه Gender ونوعه enum وجيت تكتب

Gender. [دوال] ليه وليس به دوال

س/ من أين أتت الدوال مع العلم إنك لم تكتب دالة؟

ج/ * ال Compiler بتاع جافا لما انت بتنشأ أي enum بيعمل مصفوفة اسمها Values() تحتوي على الثوابت الي انت عملتها ومرتبة حسب الترتيب الي انت عملته

Ex: Gender.Values() ⟶ دي تعتبر مصفوفة

```
for(Gender g : Gender.values())
{
    g.printString();
}
```

* راجع الدرس ٨٧ جافا ١٠١ عشان الصيغة بتاعت for

lesson 32

استخدام enum و valueOf

Gender.
valueOf(String name)

* valueOf : الدالة دي بتخليها تاخد ثابت
ع حسك نص وترجعلك الثابت الفعلي
"نفس الكائن نفسه"

Ex

```
Gender g = Gender.valueOf("welcome");
```

* نفس هي طريقة التعامل مع الثابت
بشكل نصي

* شاهد الفيديو ⇐ يُنصح باستخدام try , catch عشان لو المستخدم أدخل حاجة غير متوقعة

---

lesson 33

العلاقة بين enum , interface

* الـ enum يقدر يعمل implements لـ interface

* شاهد الفيديو

---

lesson 34

خصائص الـ enum

* بإمكانك حذف الفراغات من الـ enum مدة . مش غلط

* لكن يفضل وضعها عشان تحتفظ بالقيمة اللي بايجي من طريق الواجهة ، وتقدر توصلها عن طريق جميع الدوال.

---

lesson 35

الكلاس السمي object وفائدته

هام

* يوجد كلاس في جافا اسمه object وهو الكلاس الأبسط في جميع الكلاسات في جافا

* بمعنى أي كلاس تعمله بيكون تلقائياً مشتق من object

* trick :- هو كلاس ومكان اسمه object

* فائدته : انك لو عايز تمرر أي كائن من أي كلاس فستخدم object لأن جميع الكلاسات مشتقة منه تلقائياً .

lesson 36

* استخدام الكلاس object كباراميتر

* بينفع عادي ^^
* شوف الفيديو

---

lesson 37

استخدام كلاس object كـ Return value

* شاهد الفيديو

---

lesson 38

جلب معلومات باستخدام getclass

* يوجد في جافا كلاس يسمى Class

class | Class { }

* كل الدوال ترث دالة اسمها getclass فيها معلومات عن الكائن

```
A val = new A();
val.getclass().getName();
```

getName() → دالة بتظهر اسم الكلاس

getclass() → Return the runtime class of an object

يعني بيرجع كائن نوعه كلاس

---

lesson 39

استخدام الدالة ككائن أثناء الإرجاع

Class B

```
public A getobj()
{
  A x = new A()
  return x;
}
```

Main

```
B val = new B();
A y = val.getobj();
y.x = 10;
```

A y ← ودي كامل

y.x = 10 ← وهذا مسحوبها

البديل:

```
val.getobj().x = 10;
```

* في جافا يوجد syntax يمكنك إنك تتعامل مع القيمة الراجعة أو الكائن الراجع مباشرة بدون تخزينه

* كل دالة تمثل الكائن الذي بيرجعه

EX: Method.Method.Method.....

lesson 40

استخدام instanceOf للتعرف على نوع الكائن

* إذا عندك دالة تأخذ باراميتر نوعه Object وهيمر أكثر من كائن فلتمييز الكائنات عن بعضها وعشان تعرف الكلاس بتاعه ⇐ استخدم instanceOf

* فائدتها ممكن تعمل casting عشان تعرف تستخدم اللي انت عايزه.

* شاهد الفيديو

---

lesson 41

مقدمة في ال Nested Classes

* معناها: كلاسات بداخل بعضها البعض

* بالإمكان إنشاء كلاسات بداخل أي كلاس

* الكلاس الأساسي يسمى outer class والكلاس المُنشأ داخله يسمى inner class

---

lesson 42

أنواع ال Nested classes

* الكلاس الداخلي يأتي بنوعين: ① static ② non-static "Inner."

* في العادة يتم تسمية ال non-static ب Inner class.

* الكلاس ال static يتم كتابته هكذا ⇐ static class Ex{...}

* " " not static يتم وضع أحد ال Access Modif. مثل public ⇐ public static class Ex{...}
private
⋮

---

lesson 43

Static Nested class

* الكلاس الذي نوعه static ما يقدر يتعامل مع الأشياء التي في الكلاس الخارجي والتي تكون نوعها static

* الكلاس الذي نوعه static لا يقدر يوصل للمتغيرات والدوال التي من نوع غير static

* لتعريف كائن جديد من الكلاس الداخلي:

A. اسم الكلاس الداخلي EX = new A. اسم الكلاس الداخلي ();

(EX ↓ اسم الكائن)

مثال: A.x the_value = new A.x();

lesson 44

التعرف على Inner classes

* لا يمكن تعريف متغيرات ودوال من نوع static داخل الـ Inner class العادي

* يمكن تعريف ثابت من نوع final بداخل Inner class عادي ⇐ `public final static int y=10;`

* الـ Inner class يقدر يوصل لأي عناصر في الـ outer class حتى لو كانت private وحتى لو كانت static أو غير static

* لا يمكن إنشاء كائن من هذا النوع مباشرة بل لابد من إنشاء كائن من الكلاس الأساسي outer class أولاً ثم يتم إنشاء كائن من Inner class عن طريق كائن الـ outter class

* مثال:

```
A val = new A();
A.Y val2 = val.new Y();
```

* شاهد الفيديو

---

lesson 45

Local Inner class

* في جافا يمكن تعريف كلاس بداخل دالة، وهذا يسمى Local Inner class

* سمي كلاس محلي ⇐ لأنه لا يمكن إنشاء منه كائنات إلا بداخل الدالة اللي تعرف فيها فقط، يعني عرفته داخل هذه الدالة فقط

* شاهد الفيديو

* لا يمكن استخدام Access Modifiers مع الـ Local class لأنه معرف داخل دالة، وملوش أي صلة بالكلاس اللي برا، يعني هو داخل الدالة فقط

outter class

lesson 46

Anonymous Inner Class

+ عند كتابة كلاس جديد بداخل دالة هكذا

```
public void anotherRun() {
    new A() {};
}
```

دي معناها إنك عملت extends لـ A والدوال الجديدة هتكتب بين القوسين

+ شاهد الفيديو

ختام الدورة ^^